ПРОГРАММИРОВАНИЕ, 2024, № 3, с. 3–13

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.92

ОСОБЕННОСТИ РАЗРАБОТКИ БОРТОВОЙ СИСТЕМЫ ВИЗУАЛИЗАЦИИ ДЛЯ ГРАЖДАНСКИХ ВОЗДУШНЫХ СУДОВ

© 2024 г. Б. Х. Барладян[a], *, Н. Б. Дерябин[a], **, А. Г. Волобой[a], *, В. А. Галактионов[a], ****, Л. З. Шапиро[a], *****, И. В. Валиев[a], ******, Ю. А. Солоделов[b], *******

[a] *Институт прикладной математики им. М.В. Келдыша РАН*
125047 Москва, Миусская пл., д. 4, Россия
[b] *Государственный научно-исследовательский институт авиационных систем*
125319 Москва, ул. Викторенко, д. 7, Россия
**E-mail: bbarladian@gmail.com*
***E-mail: dek@keldysh.ru*
****E-mail: voloboy@gin.keldysh.ru*
*****E-mail: vlgal@gin.keldysh.ru*
******E-mail: pls@gin.keldysh.ru*
*******E-mail: piv@gin.keldysh.ru*
********E-mail: yasolodelov@2100.gosniias.ru*

Поступила в редакцию: 19.01.2024 г.
После доработки: 19.01.2024 г.
Принята к публикации: 24.01.2024 г.

Приборные панели современных самолетов создаются по концепции “стеклянной кабины”. Эта новая идеология интерфейса позволяет улучшить восприятие важной полетной информации за счет отображения ее на одном многофункциональном дисплее. В работе рассматриваются проблемы, возникающие при разработке сертифицируемой системы визуализации дисплея пилота, предназначенной для работы на гражданских воздушных судах под управлением российской операционной системы реального времени JetOS. В статье приведено несколько алгоритмических решений, позволяющих добиться приемлемой скорости визуализации. В частности, подробно описано решение проблемы жесткого расписания разделов операционной системы, благодаря которому удалось преодолеть деградацию скорости визуализации. Намечены пути дальнейших работ.

Ключевые слова: дисплей в кабине, бортовая система визуализации, OpenGL SC, RTOS, авионика, драйвер графического процессора

DOI: 10.31857/S0132347424030018, **EDN:** QBCOYU

1. ВВЕДЕНИЕ

Комплексы бортового оборудования современных гражданских авиалайнеров относятся к системам, критическим с точки зрения безопасности (safety critical). Они работают под управлением операционной системы реального времени (ОСРВ), необходимость использования которой вытекает из современной концепции построения авиационных систем “Интегрированная модульная авионика” [1]. Ключевой особенностью этой концепции является выполнение нескольких функциональных приложений, реализующих программное обеспечение той или иной самолетной системы, на одном процессоре. Приложения при этом должны быть разделены по времени исполнения, вытесняющая многозадачность запрещена. Такой режим работы приложений и обеспечивается ОСРВ [2, 3].

Каждое разрабатываемое гражданское воздушное судно и его комплектующие должны проходить сертификацию, что является обязательным требованием для всех систем и продуктов, критических с точки зрения безопасности. Для сертификации программного обеспечения бортового оборудования воздушного судна существует стандарт DO-178C [4], определяющий, в частности, требования к процессу разработки ПО. Эта специфика не позволяет применять готовые, известные решения для той или другой функциональности. Программный интерфейс ОСРВ, предназначенный для использования на борту

воздушного судна, должен соответствовать стандарту ARINC653 [5].

До настоящего времени зарубежные ОСРВ, такие как VxWorks 653 или Thales MACS2, применялись в качестве бортовой операционной системы на разрабатываемых отечественных воздушных судах. Но в последнее время использование зарубежных программных продуктов сталкивается с серьезными проблемами или даже становится невозможным. Это послужило причиной разработки отечественной операционной системы реального времени (ОСРВ) JetOS [3]. Процесс разработки JetOS идет в соответствии с DO-178C, и также она соответствует стандарту ARINC653 для бортовых авиационных операционных систем. Так как бортовые системы для различных самолетов строятся на разных вычислительных платформах (процессорах и соответствующих GPU), то операционная система должна быть адаптирована и сертифицирована под каждую из них.

При разработке кабин современных самолетов существует тенденция использовать большие дисплеи. Эта концепция называется “стеклянная кабина”. На один дисплей можно выводить различную информацию о полетной навигации и состоянии оборудования самолета. На рис. 1 показана приборная панель МС-21, разработанная в соответствии с этой концепцией.

Таким образом, разработка системы визуализации дисплеев пилота воздушного судна является неотъемлемой частью разработки бортовой ОСРВ. Поэтому процесс ее разработки также должен соответствовать авиационным стандартам DO-178C и ARINC653. При этом задача осложняется высокими требованиями к надежности визуализации и невысокой производительностью процессоров, так как на борт обычно ставятся энергосберегающие платформы, имеющие историю надежного использования.

Статья организована следующим образом. Раздел 2 представляет архитектуру бортовой системы визуализации и особенности двух стандартов OpenGL SC, предписанных для использования в авиации. В разделе 3 рассказывается о создании чисто программной реализации системы визуализации дисплея пилота. В разделе 4 показаны особенности реализации графической компоненты с аппаратным ускорением. Раздел 5 посвящен решению проблемы замедления визуализации из-за жесткого расписания разделов операционной системы. Раздел 6 содержит заключение.

Рис. 1. Приборная панель самолета МС-21.

2. СИСТЕМА ВИЗУАЛИЗАЦИИ В КАБИНЕ ПИЛОТА

Одним из важных требований к графической компоненте ОСРВ является использование специальной версии графической библиотеки OpenGL SC (SC – safety critical), предназначенной для систем, критических с точки зрения безопасности. В настоящее время библиотека OpenGL SC представлена двумя версиями стандарта: 1.0.1 и 2.0. Стандарт OpenGL SC1.0.1 является подмножеством стандарта OpenGL 1.3.

Стандарт OpenGL SC2.0 основан на стандарте OpenGL ES2.0, но не является его подмножеством. Он разработан для графического оборудования, работающего на встроенных платформах. В нем удалены те аспекты OpenGL ES2.0, которые не соответствуют детерминированной безопасности для критических по безопасности программных приложений. В частности, удален встроенный компилятор шейдеров. Предполагается, что вершинный (vertex) и фрагментный (fragment) шейдеры отдельно компилируются и связываются с двоичным объектом программы, который генерирует исполняемый код из скомпилированных шейдеров. При этом в стандарте не оговорен способ создания и интерфейс двоичного объекта программы. Стандартизован только интерфейс функции ProgramBinary(), которая позволяет загрузить предварительно скомпилированный двоичный объект.

Стандарты OpenGL SC2.0 и OpenGL SC1.0.1 значительно отличаются. Некоторые функции версии 1.0.1 не существуют в версии 2.0 и могут быть заменены шейдерными программами. Можно сказать, что в OpenGL SC2.0 упор делается на программируемый конвейер трехмерной графики, а в OpenGL SC1.0.1 – на фиксированную функциональность.

В общем виде систему визуализации в кабине пилота можно представить в виде схемы (рис. 2).

Графическая библиотека, написанная по одному из стандартов OpenGL SC, генерирует изображения для каждого кадра и выводит их на экран, используя драйвер дисплея.

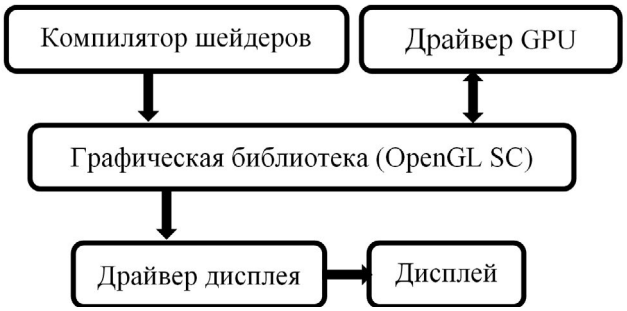

Рис. 2. Схема визуализации в кабине самолета.

При разработке программной реализации графической библиотеки, т. е. реализации, не использующей аппаратное ускорение конкретной вычислительной платформы, только драйвер дисплея должен учитывать специфику конкретного GPU. Программная версия графической библиотеки была разработана на языке C. Ее перенос на любую платформу не должен вызывать существенных затруднений.

Для использования аппаратной поддержки графического процессора для каждой вычислительной платформы необходимо разрабатывать драйвер GPU, который обеспечит доступ графической библиотеки к необходимым ресурсам графического ускорителя.

Для работы с приложениями, написанными по стандарту OpenGL SC2.0, требуется внешний компилятор шейдеров, поскольку стандарт OpenGL SC2.0 предполагает загрузку шейдеров в виде уже скомпилированного двоичного объекта.

3. ПРОГРАММНАЯ ВЕРСИЯ БИБЛИОТЕКИ OPENGL SC1.0.1

Как уже было сказано, для программной реализации графической библиотеки только драйвер дисплея должен учитывать специфику конкретной вычислительной платформы.

Разумеется, программная реализация библиотеки значительно уступает по производительности библиотеке, реализованной с использованием аппаратного ускорения. Однако у нее есть два существенных преимущества. Во-первых, сама программная версия OpenGL является платформонезависимой, а значит, может быть относительно легко перенесена на любую выбранную вычислительную платформу. Во-вторых, подготовка сертификационного пакета для программной реализации библиотеки не вызывает существенных проблем, поскольку весь код доступен и написан на языке C, а процесс его создания может быть проведен в соответствии с DO-178C.

При создании программной версии библиотеки OpenGL SC были разработаны новые алгоритмы, ускоряющие визуализацию в несколько раз. Далее, для повышения производительности мы распараллелили работу библиотеки на многоядерных процессорах, используя специальное расширение стандарта ARINC653, реализованное в JetOS. Расширение называется Asymmetric Multi-Processing (AMP). Это позволило получить дополнительное ускорение в 2.5–3 раза для типичных авиационных приложений, используя многоядерные процессоры и не отклоняясь от разрешенных стандартов. Реализация программной версии OpenGL SC1.0.1 подробно описана в работах [6, 7].

Но работа программной реализации OpenGL заканчивается генерацией изображения кадра в памяти процессора, а для вывода на экран дисплея еще необходима библиотека фрейм буфера, без которого библиотека просто не может быть использована в реальном бортовом оборудовании. И если сама программная реализация OpenGL является платформонезависимой, то драйвер дисплея, естественно, зависит от используемого оборудования. Тем не менее эта реализация существенно проще, чем полноценного драйвера GPU. Например, для платформы i.MX6 нам удалось реализовать такой драйвер с минимальным использованием кода, специфичного для GPU.

При разработке графической компоненты бортовой ОСРВ важно понимать, с какими данными она работает и какие требования предъявляются к ней. Визуализируемые приложения – это не произвольные виртуальные сцены, а типовые приложения, используемые в авиации. Главным из них является Primary flight display (PFD). На рис. 3 приведен PFD для Sukhoi Superjet 100, а на рис. 4 – для МС-21. Примером других приложений может быть индикатор состояния дверей Doors (рис. 5).

Так как типичные авиационные приложения представляют визуализацию состояния приборов и датчиков, то минимально приемлемая скорость визуализации может быть ниже, чем для обычной анимации. Уже при 15 кадрах в секунду данные достаточно оперативно отображаются на экране, картинка выглядит вполне приемлемо. Для оптимизированной программной версии библиотеки при использовании четырех ядер процессора удалось получить скорость ~14 кадров в секунду для наиболее сложного приложения SS_PFD. Остальные доступные приложения показали более высокую скорость (детали приведены в табл. 1).

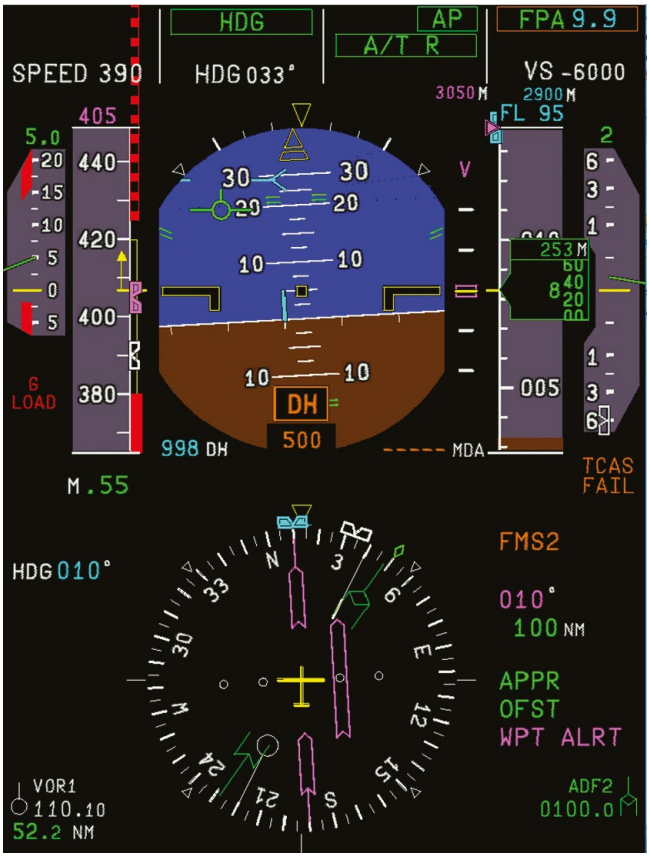

Рис. 3. Primary Flight Display SS_PFD.

Таблица 1. Скорость визуализации различных реализаций OpenGL SC в кадрах в секунду

Приложение	SWGL1	SWGL4	HWGL
SS_PFD (рис. 3)	5.9	13.8	10.8
MC_PFD (рис. 4)	6.3	15.6	20.0
Doors (рис. 5)	12.0	34.7	60
Doors + PDF (рис. 6)	4.6	6.2	16.5

Использование больших дисплеев в кабине пилота подразумевает их многооконность, чтобы различная информация была оперативно доступна пилоту. Реализация многооконной визуализации с использованием программной версии OpenGL SC позволила получить скорость 6.2 кадра в секунду для приложений (рис. 6). Детали многооконной визуализации описаны в [7].

Разработанная нами программная реализация библиотеки OpenGL SC1.0.1 при использовании многоядерности процессора оказалась достаточно эффективной для многих практических авиационных приложений, но не для всех. Кроме того, в ряде случаев возникает необходимость использования ядер процессора для других компонент комплекса бортового оборудования. В связи

Рис. 4. Primary Flight Display MC_PFD.

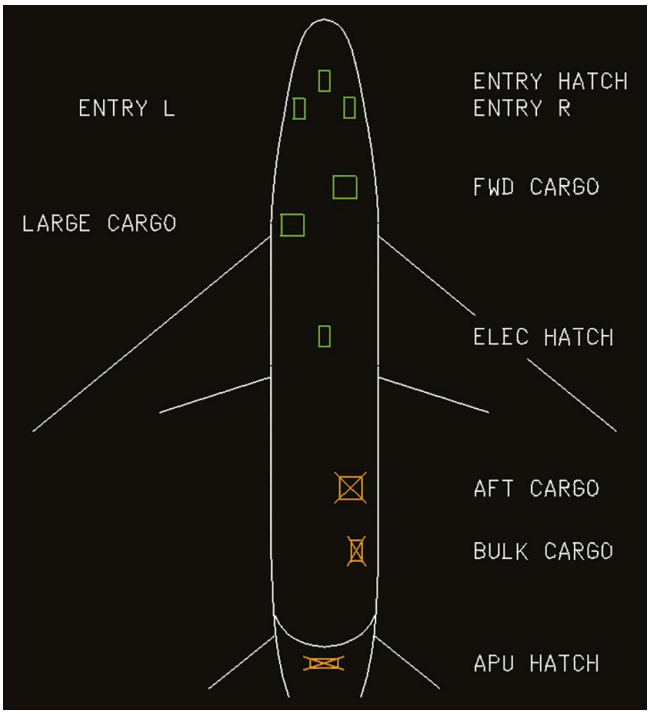

Рис. 5. Индикатор состояния дверей Doors.

с этим вопрос использования GPU для ускорения визуализации остается актуальным, тем более, что именно для этих целей GPU в основном и разрабатывались.

4. РЕАЛИЗАЦИЯ БИБЛИОТЕКИ OPENGL SC С АППАРАТНЫМ УСКОРЕНИЕМ

Реализация различных версий OpenGL SC на базе коммерческих драйверов GPU рассматривается в работах [8–10]. Однако необходимость разработки сертификационного пакета для графической компоненты в соответствии с требованиями КТ-178С требует доступа ко всем ее кодам. В большинстве случаев драйверы, предоставляемые производителями GPU, не имеют открытого

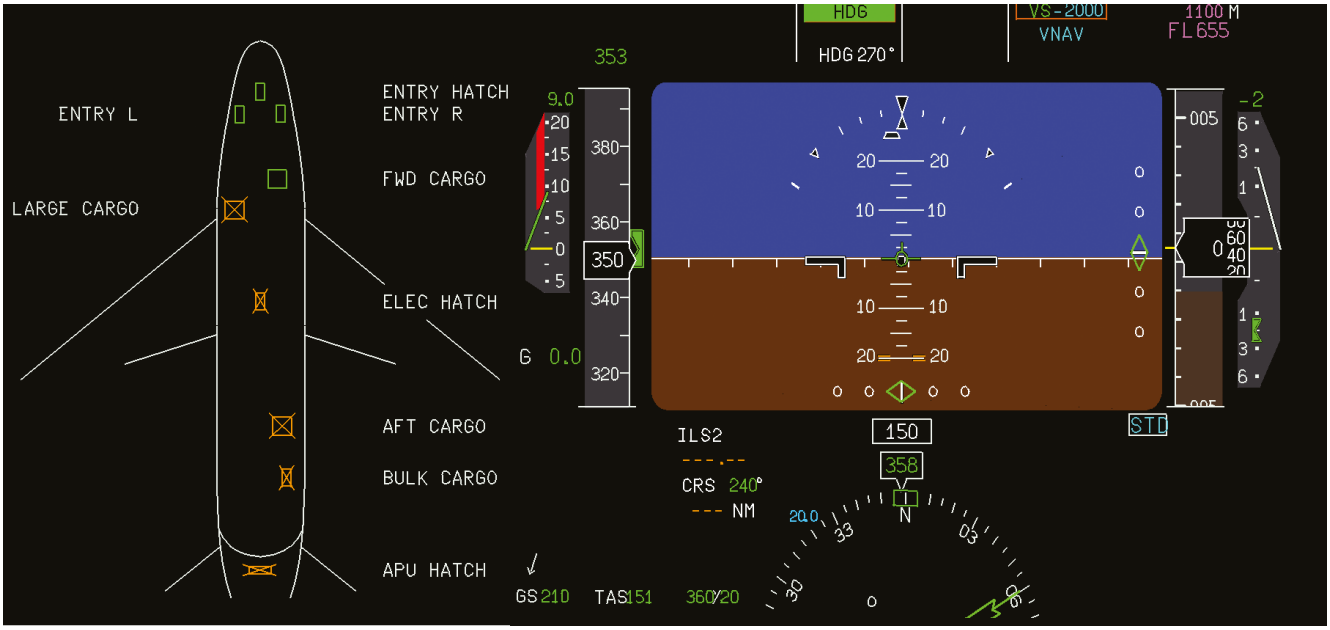

Рис. 6. Многооконная визуализация Doors и PFD.

программного кода, поэтому такое ПО не может быть сертифицировано для использования в авиации. Также драйвер необходимо адаптировать для работы под управлением ОСРВ JetOS. Поэтому наша разработка базировалась на пакете Mesa 3D [11].

Mesa 3D представляет собой программную реализацию с открытым исходным кодом различных спецификаций графического API, включая OpenGL, Vulkan и др. Mesa 3D переводит эти спецификации в драйверы конкретного графического оборудования. Некоторые производители процессоров, такие как, например, AMD или Intel, сами разрабатывают драйверы для этого пакета с открытым кодом для производимых ими процессоров. Другие производители, такие как Nvidia или Vivante, полностью заменяют драйверы в Mesa 3D, обеспечивая собственную реализацию библиотеки OpenGL. Для такого оборудования сообщество разработчиков Mesa создает альтернативные открытые драйверы методом обратной разработки (reverse engineering), такие как Nouveau для Nvidia или Etnaviv для Vivante. При использовании платформы i.MX6 с графическим процессором Vivante единственной возможностью разработки сертифицируемой библиотеки OpenGL SC является разработка драйвера на основе Etnaviv.

Детально процесс реализации графической библиотеки OpenGL SC с аппаратной поддержкой на базе пакета Mesa 3D описан в работах [12, 13]. Отметим здесь, что реализация многооконного режима потребовала разработать принципиально другой алгоритм по сравнению с программной версией библиотеки, поскольку при использовании аппаратной поддержки может быть использован только один экземпляр OpenGL. Аппаратная поддержка GPU позволила в большинстве случаев увеличить скорость визуализации и для стандартных приложений, и для многооконной визуализации (см. табл. 1). В таблице SWGL1 – программная версия библиотеки при использовании одного ядра процессора, SWGL4 – программная версия библиотеки при использовании четырех ядер процессора, HWGL – версия библиотеки с аппаратной поддержкой.

В реальности при распараллеливании графической компоненты с программной версией OpenGL SC в многооконном режиме было задействовано только три ядра процессора: два ядра для каждой составляющей изображения (Doors и PFD) и одно ядро для компоновщика всего изображения.

Отдельно следует отметить реализацию графической библиотеки, поддерживающей стандарт OpenGL SC2.0, который в настоящее время является наиболее используемым в авиационных приложениях. Реализация программной версии этого стандарта не имеет практического смысла, поскольку шейдеры в настоящее время эффективно реализованы в современных GPU, а их программная реализация будет неэффективна.

Но для работы с приложениями, написанными по этому стандарту, необходима реализация внешнего компилятора шейдеров, поскольку стандарт OpenGL SC2.0 предполагает загрузку

шейдеров в виде уже скомпилированного двоичного объекта. Код компилятора шейдеров в пакете Mesa 3D написан на языке C++, который практически невозможно сертифицировать по авиационным нормам. Получается, что этот код трудно исключить для стандарта 1.0.1, но относительно просто для стандарта 2.0, поскольку компилятор шейдеров вынесен во внешнюю программу, которая используется только на этапе подготовки приложений. Таким образом, сертификация реализации библиотеки OpenGL SC стандарта 2.0 с аппаратной поддержкой получается значительно проще, чем стандарта 1.0.1.

4.1. Проблемы использования аппаратного ускорения

В некоторых случаях использование аппаратного ускорения может привести не к ускорению, а к замедлению скорости визуализации. Это видно на примере приложения SS_PFD: в табл. 1 скорость визуализации с аппаратной поддержкой ниже, чем чисто программной. Это часто связано с тем, что приложения для бортового оборудования создаются в соответствии со стандартами и с помощью автоматизированных систем, которые не учитывают специфику GPU. Более серьезная проблема возникла (и была решена нами) с приложением, созданным по стандарту ARINC661 [14].

Система отображения в кабине экипажа Cockpit display system (CDS) предоставляет видимую и звуковую информацию о воздушном судне и окружающей среде и получает команды управления от экипажа. Через нее экипаж управляет современным самолетом. Интерфейсы этой системы должны удовлетворять авиационному стандарту ARINC661. Некоторые приложения, взаимодействующие с CDS, создаются с помощью программ автоматизированного проектирования, гарантирующих соответствие этому стандарту [15].

Приложение, представляющее карту движения по аэродрому с взлетно-посадочной полосой и рулежными дорожками (рис. 7), использует сервер ARINC661, разработанный компанией Ansys Inc. [15].

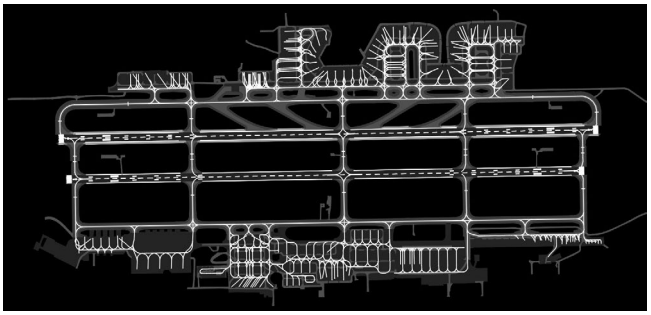

Рис. 7. Карта рулежных дорожек аэродрома.

Приложение оказалась неэффективным с точки зрения использования графической компоненты с аппаратным ускорением. Оно визуализировало каждый отрезок линии и каждый треугольник треугольной сетки отдельным набором команд библиотеки OpenGL, в то время как эффективный интерфейс GPU подразумевает выполнение как можно большого набора команд за одно обращение. В работе [16] мы оптимизировали работу сервера ARINC661 путем добавления в конвейер визуализации препроцессора, который объединяет команды в группы рисования однородных примитивов, отрезков или треугольников. Само приложение при этом не модифицировалось. В результате этой оптимизации визуализация карты аэродрома (см. рис. 7) была ускорена с 1.7 до 15.2 кадров в секунду.

4.2. Подготовка OpenGL SC с аппаратной поддержкой к сертификации

Как уже говорилось, прототип библиотеки был разработан на основе пакета Mesa 3D. В соответствии со стандартом DO-178C в ПО бортового оборудования не должно быть неисполняемого кода. Поэтому одной из важных задач было исключение кода, который не будет использоваться в OpenGL SC. Это сложная задача в силу практического отсутствия документации в пакете Mesa 3D и большого объема сложного кода, который предполагает выполнение под управлением различных операционных систем и использование различных платформ и GPU.

Для этой цели был использован итерационный процесс, на каждом шаге которого достигается частичное снижение сложности и избыточности пакета. Сначала использовались, в основном, формальные методы – это методы, которые не требуют понимания работы кода, а только понимания общих принципов программирования. Простейшим применением формального метода является удаление неиспользуемой функции. Если какая-то функция не используется, ее описание и определение удаляются. После удаления неиспользуемой функции могут появиться другие неиспользуемые функции. В определении их помогает компилятор, который диагностирует большинство неиспользуемых статических функций.

Аналогично, можно удалить неиспользуемую переменную или неиспользуемый член структуры. Переменная не используется, если она нигде не читается. Может оказаться, что переменной

(члену структуры) присваивается только одно значение. В этом случае ее тоже можно удалить, заменив чтение этой переменной ее значением.

Если в процессе адаптации функция стала пустой, ее и ее вызовы можно удалить. Если она стала возвращать единственное значение, то можно заменить вызовы этой функции данным значением. Если в файле не осталось внешних функций и переменных, то он удаляется, и т. д.

Для анализа неиспользуемого кода оказалась полезным использование такой функциональности как получение покрытия кода при работе приложения. Эта функциональность [17] была разработана и обычно применяется при тестировании программного обеспечения. Ее основное назначение – определение насколько хорошо разработанные тесты покрывают код, его функции и ветвления. Использование этой функциональности оказалась эффективно для исключения неиспользуемого кода библиотеки.

Методы, использованные в этом итерационном процессе, подробно описаны в работе [13]. В результате удалось на порядок снизить объем кода.

5. ПРОБЛЕМЫ РЕАЛЬНОГО ВРЕМЕНИ И ЖЕСТКОГО РАСПИСАНИЯ

Каждое программное приложение, работающее на борту гражданского судна, обязано следовать стандарту ARINC653, где прописаны жесткие требования безопасности. Для обеспечения реального времени система должна иметь возможность достаточно часто переключать использование процессора для выполнения разных задач, поэтому невозможно использовать все ресурсы процессора только для графической компоненты. В соответствии со стандартом каждое приложение выполняется в отдельном разделе (partition) ОСРВ. Диспетчеризация работы разделов является строго детерминированной по времени. В зависимости от конфигурации разделов в модуле, общих требований к ресурсам, наличия ресурсов и требований отдельных разделов формируется расписание времени активации разделов, в котором отдельным разделам выделяются их окна. Каждый раздел получает процессорное время в соответствии с этим расписанием.

Для обеспечения реального времени операционной системы необходимо, чтобы активация соответствующего раздела происходила не реже чем через заданный интервал. Как правило, время реакции системы не должно превышать десятков, максимум сотни миллисекунд. Кроме графического приложения, которое отображает поступающую информацию, в системе работают другие приложения. Как минимум должно быть еще одно приложение, которое обрабатывает поступающую информацию и передает ее графическому приложению. Таким образом, длительность окна раздела, в котором выполняется графическое приложение, не может превышать нескольких десятков миллисекунд (желательно не более 20 мс). Следует отметить, что переключение разделов – довольно затратная процедура. Требования по безопасности регламентируют, что при каждом переключении должны быть очищены кэши и регистры, сохранено их содержимое и затем восстановлено при продолжении работы данного раздела. Поэтому полезное процессорное время для каждого приложения меньше, чем время выделенного раздела, как это показано на рис. 8, где полезное время в каждом разделе представлено закрашенным прямоугольником. В результате получается, что при работе дополнительных разделов скорость визуализации снижается.

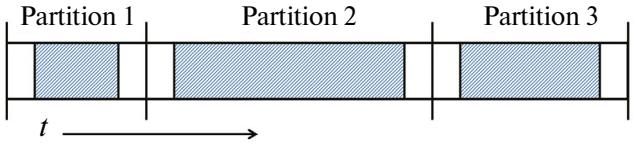

Рис. 8. Расписание выполнения разделов в ОСРВ. Закрашенные прямоугольники соответствуют полезному времени выполнения приложения, пустые прямоугольники в начале и конце каждого раздела – сохранение и восстановление кэшей и регистров.

5.1. Схема работы алгоритма

Для решения этой проблемы мы выделили работу библиотеки OpenGL SC в отдельный раздел ОСРВ (сервер OpenGL). Она выполняется на отдельном ядре процессора с учетом всех ограничений, налагаемых стандартом. Из этого же раздела организовано взаимодействие с GPU. Этот подход является дальнейшим развитием идей, предложенных в работах [13, 16]. В работе [13] драйвер OpenGL, использующий GPU, работал в отдельном разделе и принимал команды от других приложений, но все разделы для различных окон работали на одном ядре процессора. В работе [16] драйвер OpenGL работал в отдельном разделе на отдельном ядре, но при этом использовалось сжатие команд, специфичное для сервера ARINC661, которое будет неэффективно и не всегда возможно в общем случае.

В общем случае разработанный алгоритм позволяет обеспечить эффективную реализацию вывода результатов работы различных

графических приложений на дисплей пилота [13]. Приложения при этом могут работать в различных разделах ОСРВ JetOS и на различных ядрах процессора. В частности, алгоритм обеспечивает эффективную реализацию многооконного режима. В настоящей работе исследовалась только эффективность использования предложенной схемы алгоритма для работы в режиме жесткого расписания.

Мы реализовали драйвер OpenGL общего вида, который работает на отдельном ядре процессора и обрабатывает команды OpenGL, поступающие из графических приложений, которые работают на других ядрах процессора. Подход обеспечивает определенное распараллеливание работы графического приложения. Часть работы (подготовка данных и команд) происходит на одном ядре процессора, а другая часть (сервер OpenGL) – на другом ядре процессора и на GPU. Пока данный кадр визуализируется на GPU, раздел на другом ядре может подготавливать данные и команды для следующего кадра.

Схема работы алгоритма, при котором сервер OpenGL выполняется на отдельном ядре процессора, показана на рис. 9. Основное графическое приложение, использующее для рендеринга библиотеку OpenGL, представлено как Application 1. Application 2 – это дополнительное приложение, которое эмулирует выполнение других необходимых задач параллельно с основным графическим приложением. Оба приложения работают в одном модуле на одном ядре процессора с разделением по времени. Каждое из них реализовано в своем разделе JetOS и работает согласно расписанию. В наших исследованиях это приложение использовалось для моделирования загрузки модуля неграфическими задачами. В тестах оно не выполняло никакой содержательной работы, но в расписании на него отводился определенный квант времени.

5.2. Алгоритм использования сервера OpenGL

Поскольку сервер OpenGL и графическое приложение работают независимо друг от друга на разных ядрах процессора в двух отдельных копиях ОСРВ JetOS, то их работу необходимо синхронизировать. Для простоты изложения мы опустим здесь специфические для JetOS процедуры запуска отдельных копий JetOS и запуска разделов в них. Опишем только передачу данных между графическим приложением и OpenGL сервером и синхронизацию их работы.

Графическое приложение только подготавливает необходимые данные и команды в соответствии с заданным стандартом OpenGL. Мы использовали стандарт OpenGL SC2.0. Для передачи данных и команд между приложением и сервером были реализованы две специальные библиотеки **OGLOUT** и **OGLIN**, которые используют общую память для приложения и сервера. Первая библиотека заменяет (эмулирует) все команды стандарта OpenGL SC2.0 и используется в графическом приложении вместо реальной библиотеки OpenGL. Вместо выполнения соответствующих команд она просто записывает все необходимые данные и идентификаторы команд в специальные массивы, расположенные в общей памяти (Buffer). Библиотека **OGLIN** используется в OpenGL-сервере. С ее помощью сервер читает информацию, записанную в буфере, и последовательно выполняет команды OpenGL, подготовленные графическим приложением.

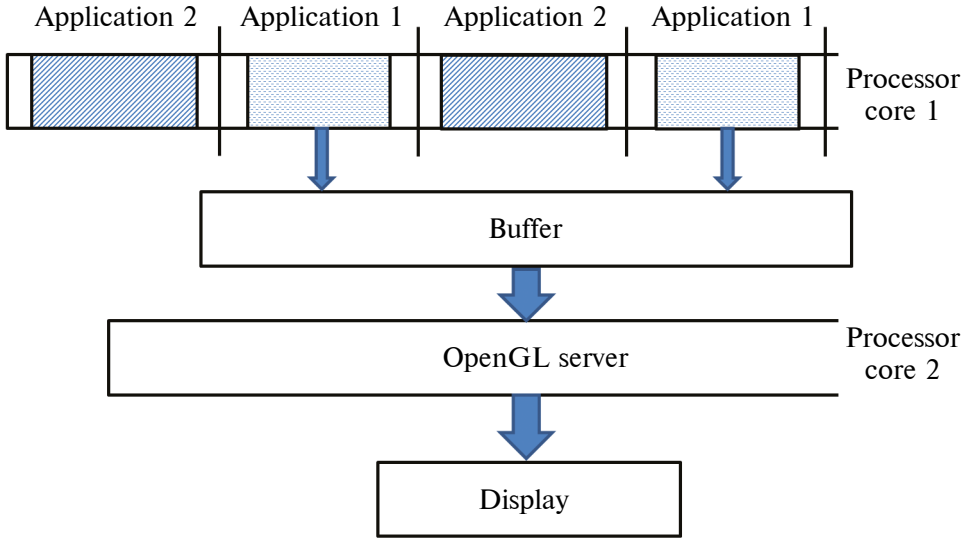

Рис. 9. Схема работы графического приложения в режиме жесткого расписания.

Синхронизация работы приложения и сервера OpenGL выполняется с помощью специальных объектов, называемых событиями, которые реализуются через небольшие общие блоки памяти между модулями. Доступ к этим объектам реализован с помощью атомарных операций (atomic operations). В нашем случае используются два события:

StartRend – устанавливается в сигнальное состояние в приложении в библиотеке **OGLOUT**, когда данные, хранящиеся в общей памяти, готовы к обработке сервером OpenGL, т. е. записаны все данные и команды, необходимые для генерации данного кадра;

EndRend – устанавливается в сигнальное состояние сервером OpenGL, когда закончена обработка заданного фрагмента данных, хранящегося в общей памяти, данный кадр выведен на экран и сервер OpenGL готов обрабатывать следующий кадр.

Первоначально событие **StartRend** устанавливается в несигнальное состояние, а **EndRend** – в сигнальное состояние.

5.3. Результаты тестов

В качестве графического приложения в тестах использовалось два приложения: SS_PFD (см. рис. 3) и более простой прототип PFD. В первом тесте Application 1 и Application 2 работали в одном модуле, но в разных разделах, при этом OpenGL использовалась непосредственно из графического приложения. Длительность окна для работы раздела Application 1 (графическое приложение) была зафиксирована в 10 мс, а длительность окна для работы раздела второго приложения варьировалась от 0 до 20 мс. Длительность 0 означает, что раздел со вторым приложением не использовался. Результаты первого теста для разных вариантов длительности второго, неграфического, приложения приведены в табл. 2.

Из табл. 2 видно, что работа второго приложения в том же модуле с разделением времени с графическим приложением заметно уменьшает скорость визуализации. Степень замедления зависит от расписания и специфики приложения, но в любом случае оно достаточно существенно.

Таблица 2. Скорости визуализации в кадрах в секунду в зависимости от длительности окна второго приложения (одно ядро процессора)

Приложение	0 mc	3 mc	5 mc	10 mc	20 mc
SS_PFD	20	16.5	15	13.3	8.9
proto-pfd	30	27	27	20	13.4

Таблица 3. Скорости визуализации в кадрах в секунду в зависимости от длительности окна второго приложения (с сервером OpenGL)

Приложение	0 mc	30 mc	40 mc	50 mc
SS_PFD	20	20.0	20.0	16.7
proto-pfd	30	30.0	26.8	24.2

Во втором тесте библиотека OpenGL была реализована в виде сервера, как показано на рис. 9. Те же два приложения по-прежнему работали в одном модуле в двух разделах, а библиотека OpenGL – в отдельном модуле на отдельном ядре процессора. Длительность окна для работы раздела графического приложения была также зафиксирована в 10 мс, а длительность окна для работы второго приложения варьировалась от 10 до 50 мс. Результаты второго теста для разных вариантов длительности второго, неграфического, приложения приведены в табл. 3.

Из табл. 3 видно, что работа второго приложения с разделением времени с графическим приложением при использовании сервера OpenGL также уменьшает скорость визуализации, но степень замедления значительно меньше. До длительности раздела второго приложения 30 мс (т. е. в три раза больше, чем длительность раздела графического приложения) замедления практически не видно. Оно начинает проявляться, только если длительность раздела второго приложения существенно превышает время, отводимого в расписании для раздела графического приложения.

6. ЗАКЛЮЧЕНИЕ

Создание бортовой системы визуализации для кабины пилота гражданского воздушного судна является сложной и многофакторной задачей. Главным требованием остается безопасность полетов. Это накладывает принципиальные ограничения, значительно усложняющие создание графической системы. Первое ограничение – это надежная, энергосберегающая вычислительная платформа. На борт воздушного судна не ставят современные и достаточно мощные графические процессоры Nvidia или AMD из-за их высоких энергетических затрат и недостаточной наработки на отказ. Надежность процессоров проверяется со временем, поэтому на борт ставятся не самые новые процессоры, производительность которых невысока по современным меркам. Вторым фактором разработки системы визуализации

является ее использование под операционной системой реального времени, которая во многом отличается от общепринятых ОС Linux или Windows. Сложность создания графической системы также связана с необходимостью ее сертификации как технологии, критической с точки зрения безопасности. Процесс ее создания должен удовлетворять жестким стандартам, предъявляемым к программному обеспечению комплекса бортового оборудования гражданского воздушного судна. Поэтому невозможно использование готовых существующих реализации графических библиотек и драйверов. При этом должна быть гарантирована приемлемая скорость визуализации авиационных приложений.

В процессе создания графической компоненты многооконной визуализации дисплея пилота нами было разработано множество алгоритмов и подходов, позволивших визуализировать авиационные приложения с приемлемой скоростью. Особенностью реализации также является работа компоненты под ОСРВ с жестким расписанием. Надо сказать, что некоторые из решений, успешно работающих для одной вычислительной платформы, могут не иметь эффекта для другой. Также авиационные приложения становятся все более сложными, что повышает безопасность полетов, предоставляя пилотам дополнительную информацию. Поэтому дальнейшие разработки будут связаны с повышением производительности графической компоненты и адаптацией к новым платформам.

СПИСОК ЛИТЕРАТУРЫ

1. *Федосов Е.А., Косьянчук В.В., Сельвесюк Н.И.* Интегрированная модульная авионика // Радиоэлектронные технологии. 2015. № 1. С. 66–71.
2. *Федосов Е.А., Ковернинский И.В., Кан А.В., Солоделов Ю.А.* Применение операционных систем реального времени в интегрированной модульной авионики. OSDAY2015. http://osday.ru/solodelov.html
3. *Солоделов Ю.А., Горелиц Н.К.* Сертифицируемая бортовая операционная система реального времени JetOS для российских проектов воздушных судов // Труды ИСП РАН. 2017. Т. 29. № 3. С. 171–178. https://doi.org/10.15514/ISPRAS-2017-29(3)-10
4. DO-178C Software Considerations in Airborne Systems and Equipment Certification. http://www.rtca.org/store_product.asp?prodid=803
5. Avionics application software standard interface (ARINC653). SAE-ITC, 2015. https://aviation-ia.sae-itc.com/standards/arinc653p0-3-653p0-3-avionics-application-software-standard-interface-part-0-overview-arinc-653
6. *Барладян Б.Х., Волобой А.Г., Галактионов В.А., Князь В.В., Ковернинский И.В., Солоделов Ю.А., Фролов В.А., Шапиро Л.З.* Эффективная реализация OpenGL SC для авиационных встраиваемых систем // Программирование. 2018. № 4. С. 3–10. https://doi.org/10.31857/S013234740000519-5
7. *Барладян Б.Х., Шапиро Л.З., Малачиев К.А., Хорошилов А.И., Солоделов Ю.А., Волобой А.Г., Галактионов В.А., Ковернинский И.В.* Система визуализации для авиационной ОС реального времени JetOS // Труды Института системного программирования РАН. 2020. Т. 32. № 1. С. 57–70. https://doi.org/10.15514/ISPRAS-2020-32(1)-3
8. *Baek N. and Lee H.* OpenGL ES1.1 Implementation Based on OpenGL // Multimedia Tools and Applications. V. 57. No. 3 (2012). P. 669–685.
9. *Baek N., Lee H.* OpenGL SC Implementation over an OpenGL ES1.1 Graphics Board // 2012 IEEE International Conference on Multimedia & Expo Workshops (ICMEW 2012). P. 671–671. https://doi.org/10.1109/ICMEW.2012.127
10. *Baek N. and Kim K.J.* Design and implementation of OpenGL SC2.0 rendering pipeline // Cluster Computing (2019). 22: S931–S936. https://doi.org/10.1007/s10586-017-1111-1
11. The Mesa 3D Graphics Library. https://www.mesa3d.org/
12. *Barladian B. Kh., Deryabin N.B., Voloboy A.G., Galaktionov V.A., Shapiro L.Z.* High speed visualization in the JetOS aviation operating system using hardware acceleration // CEUR Workshop Proceedings. 2020. V. 2744. P. 107:1–107:9. https://doi.org/10.51130/graphicon-2020-2-4-3
13. *Barladian B.K., Deryabin N.B., Shapiro L.Z., Solodelov Yu.A., Voloboy A.G. and Galaktionov V.A.* Multiwindow Rendering on a Cockpit Display Using Hardware Acceleration // Programming and Computer Software. 2021. V. 47. № 6. P. 457–465. https://doi.org/10.1134/S0361768821060025
14. ARINC Standards. https://www.aviation-ia.com/products/661p1-8-cockpit-display-system-interfaces-user-systems-part-1-avionics-interfaces-basic
15. Ansys SCADE Solutions for ARINC661 Compliant Systems, 2021. https://www.ansys.com/products/embedded-software/solutions-for-arinc-661
16. *Barladian B.K., Shapiro L.Z., Deryabin N.B., Solodelov Yu.A., Voloboy A.G. and Galaktionov V.A.* Efficient Rendering for the Cockpit Display System Designed in Compliance with the ARINC661 Standard // Programming and Computer Software. 2022. V. 48. № 3. P. 147–154. https://doi.org/10.1134/S0361768822030021
17. *Brian Gough.* An Introduction to GCC – for the GNU compilers gcc and g++ – Coverage testing with gcov. https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_81.html

SPECIFICS OF THE DEVELOPMENT OF AN ON-BOARD VISUALIZATION SYSTEM FOR CIVIL AIRCRAFTS

© 2024 B. Kh. Barladian[a], N. B. Deryabin[a], I. V. Valiev[a], A. G. Voloboy[a], V. A. Galaktionov[a], L. Z. Shapiro[a], Yu. A. Solodelov[b]

[a] *Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Miusskaya pl. 4, Moscow, 125047 Russia*

[b] *State Scientific Research Institute of Aviation Systems, ul. Viktorenko 7, Moscow, 125319 Russia*

The instrument panels of modern aircraft are created using the "glass cockpit" concept. This new interface philosophy improves the perception of important flight information by displaying it on a single multi-function display. The paper considers the problems that arise when developing a certified pilot display visualization system designed for operation on civil aircraft under the Russian real-time operating system JetOS. The paper presents several algorithmic solutions that allow achieving acceptable visualization speed. In particular, a solution to the problem of rigid scheduling of operating system partitions is described in detail. This solution allows to overcome the degradation of rendering speed. Directions for further work have been outlined.

Keywords: cockpit display, on-board visualization system, OpenGL SC, RTOS, avionics, GPU driver

REFERENCES

1. *Fedosov E.A., Kos'yanchuk V.V., Sel'vesyuk N.I.* Integrated modular avionics // Radioelektron. techn. 2015. № 1. P. 66–71.
2. *Fedosov E.A., Koverninskiy I.V., Kan A.V., Solodelov Y.A.* Application of real-tim oerating systems in integrated modular avionics. OSDAY2015. http://osday.ru/solodelov.html
3. *Solodelov Yu.A. and Gorelits N.K.* Certifiable onboard real-time operation system JetOS for Russian aircrafts design // Proceedings of the Institute for System Programming of the RAS. 2017. V. 29. № 3. P. 171–178. https://doi.org/10.15514/ISPRAS-2017-29(3)-10
4. DO-178C Software Considerations in Airborne Systems and Equipment Certification. http://www.rtca.org/store_product.asp?prodid=803
5. Avionics application software standard interface (ARINC653). SAE-ITC, 2015. https://aviation-ia.sae-itc.com/standards/arinc653p0-3-653p0-3-avionics-application-software-standard-interface-part-0-overview-arinc-653
6. *Barladian B.Kh., Voloboy A.G., Galaktionov V.A., Knyaz' V.V., Koverninskii I.V., Solodelov Yu.A., Frolov V.A., Shapiro L.Z.* Efficient Implementation of OpenGL SC for Avionics Embedded Systems // Programming and Computer Software. 2018. V. 44. № 4. P. 207–212. https://doi.org/10.1134/S0361768818040059
7. *Barladyan B.H., Shapiro L.Z., Mallachiev K.A., Khoroshilov A.V., Solodelov Yu.A., Voloboy A.G., Galaktionov V.A., Koverninsky I.V.* Rendering System for the Aircraft Real-Time OS JetOS // Proceedings of the Institute for System Programming of the RAS. 2020. V. 32. № 1. P. 57–70. https://doi.org/10.15514/ISPRAS-2020-32(1)-3
8. *Baek N. and Lee H.* OpenGL ES1.1 Implementation Based on OpenGL // Multimedia Tools and Applications. V. 57. No. 3 (2012). P. 669–685.
9. *Baek N., Lee H.* OpenGL SC Implementation over an OpenGL ES1.1 Graphics Board // 2012 IEEE International Conference on Multimedia & Expo Workshops (ICMEW 2012). P. 671–671. https://doi.org/10.1109/ICMEW.2012.127
10. *Baek N. and Kim K.J.* Design and implementation of OpenGL SC2.0 rendering pipeline // Cluster Computing (2019). 22: S931–S936. https://doi.org/10.1007/s10586-017-1111-1
11. The Mesa 3D Graphics Library. https://www.mesa3d.org/
12. *Barladian B.Kh., Deryabin N.B., Voloboy A.G., Galaktionov V.A., Shapiro L.Z.* High speed visualization in the JetOS aviation operating system using hardware acceleration // CEUR Workshop Proceedings. 2020. V. 2744. P. 107:1–107:9. https://doi.org/10.51130/graphicon-2020-2-4-3
13. *Barladian B.K., Deryabin N.B., Shapiro L.Z., Solodelov Yu.A., Voloboy A.G. and Galaktionov V.A.* Multiwindow Rendering on a Cockpit Display Using Hardware Acceleration // Programming and Computer Software. 2021. V. 47. № 6. P. 457–465. https://doi.org/10.1134/S0361768821060025
14. ARINC Standards. https://www.aviation-ia.com/products/661p1-8-cockpit-display-system-interfaces-user-systems-part-1-avionics-interfaces-basic
15. Ansys SCADE Solutions for ARINC661 Compliant Systems, 2021. https://www.ansys.com/products/embedded-software/solutions-for-arinc-661
16. *Barladian B.K., Shapiro L.Z., Deryabin N.B., Solodelov Yu.A., Voloboy A.G. and Galaktionov V.A.* Efficient Rendering for the Cockpit Display System Designed in Compliance with the ARINC661 Standard // Programming and Computer Software. 2022. V. 48. № 3. P. 147–154. https://doi.org/10.1134/S0361768822030021
17. *Brian Gough.* An Introduction to GCC – for the GNU compilers gcc and g++ – Coverage testing with gcov. https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_81.html